Math 405 Assignment 5

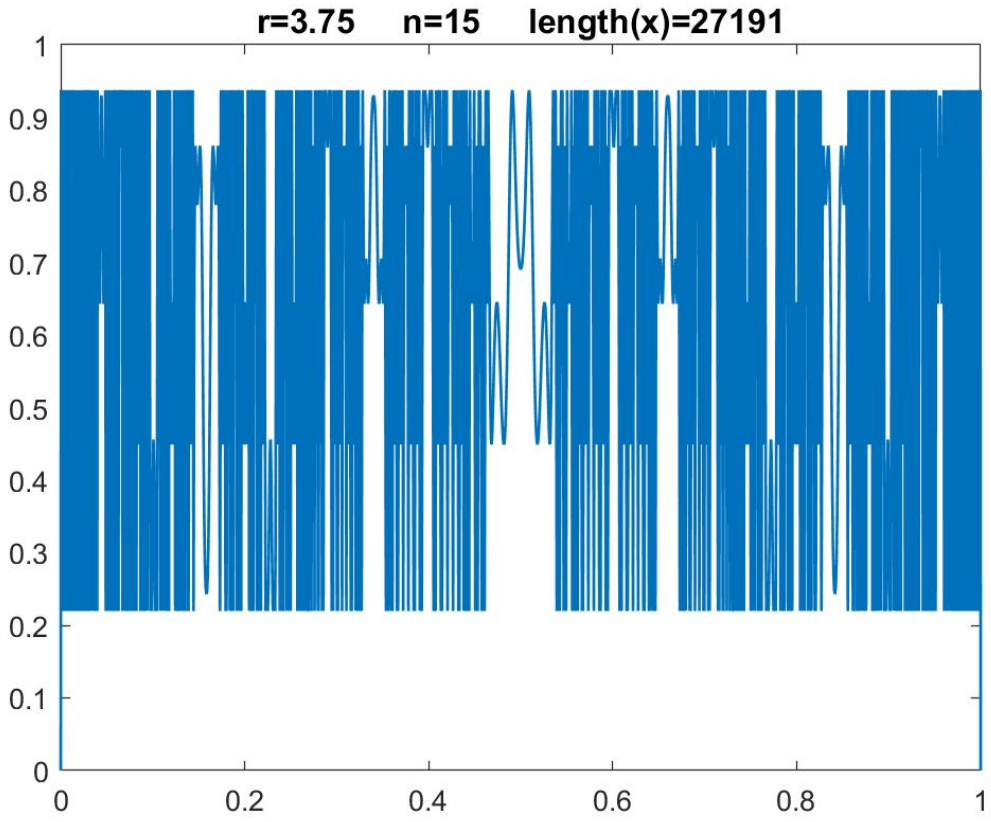
Kaan Yolsever 20569159

November 29^{th} 2018

1 Introduction

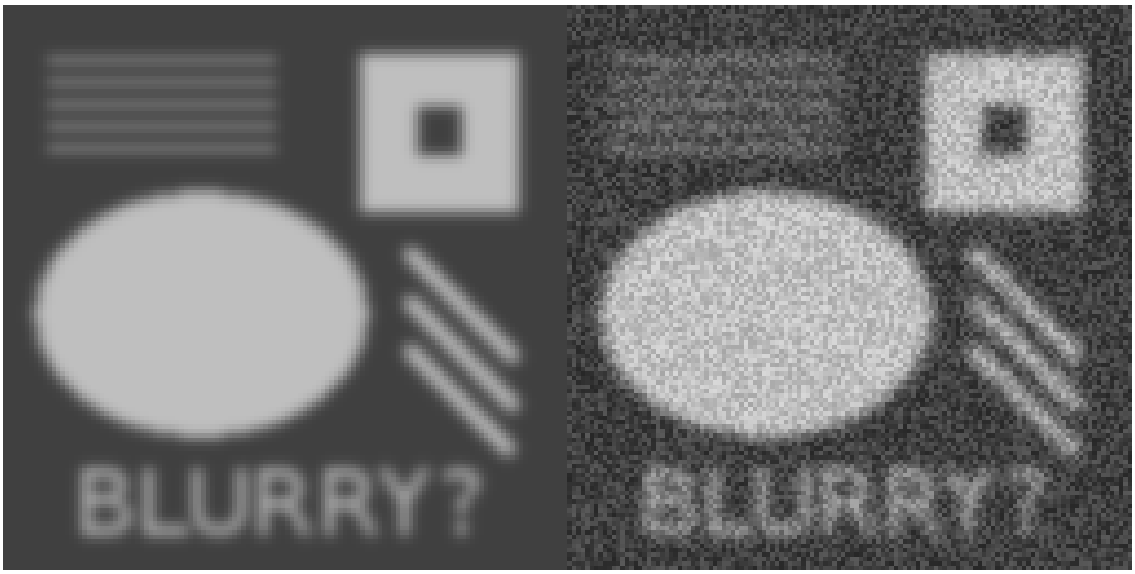
I will include the code at the end of each section.

1. I modified the logistic map code, the values of n and r , and here is the output:




```
tic
r = 3.75; x0 = chebfun('x',[0 1]);
n = 15; x = x0;
for k = 1:n, x = r*x.*(1-x); end
LW = 'linewidth'; FS = 'fontsize';
plot(x,LW,1)
ss = sprintf('r=%4.2f     n=%d     length(x)=%d', r, n, length(x));
title(ss,FS,12), axis([0 1 0 1])
```

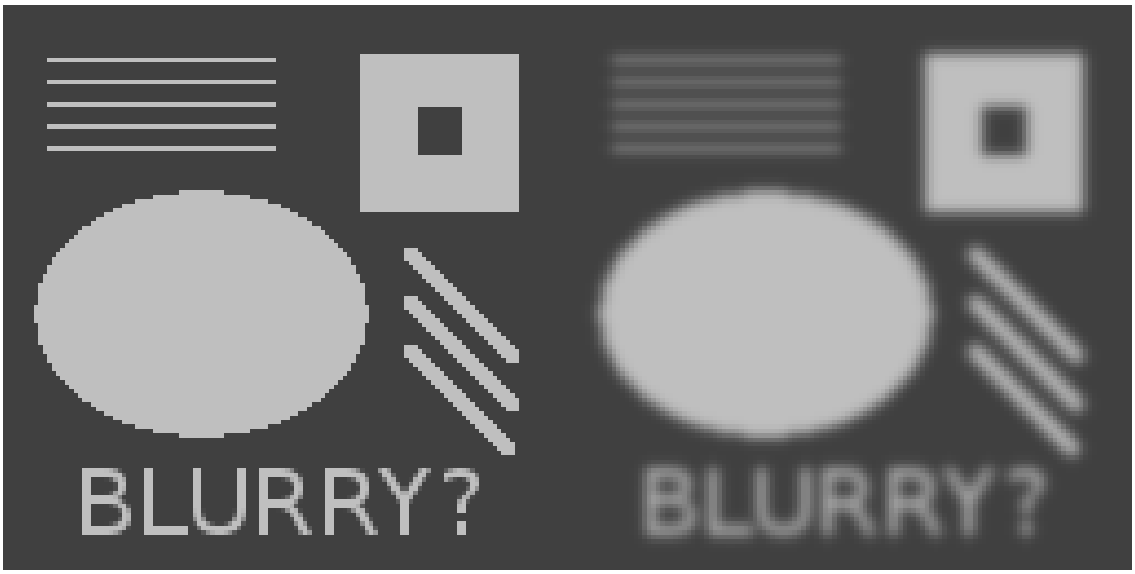
2. (a) The output is given below



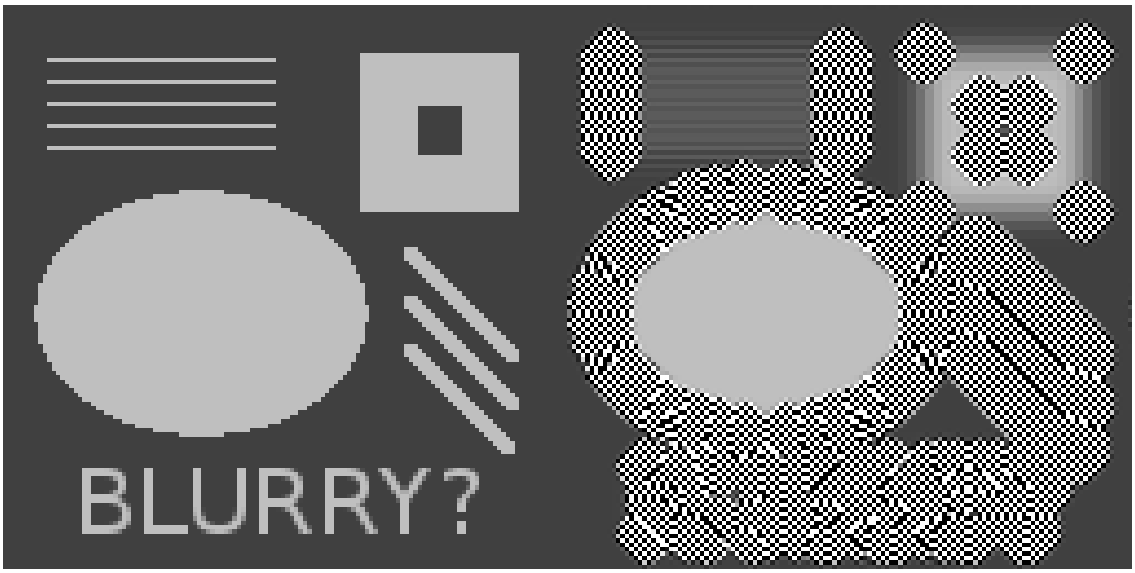
(b)
```
%% Read an image from a file
clear all
u = imread('testpat_noblur.png');
% convert image to double and scale to [0,1]
u = double(u) / 255;

[n,n2] = size(u);
if (n ~= n2)
  error('by default, this only supports square images')
end

figure(1); clf;
%pcolor(u); % try this one too
imagesc(u);
caxis([0 1])
colormap(gray)
axis equal, axis tight

% Grid and initial data:
hx = 1;
hy = 1;
x1d = 0:hx:(n-1);
y1d = 0:hy:(n2-1);
k = 0.1;    % time step (try .5, what happens?)

[x, y] = meshgrid(x1d, y1d);

% initial condition (must be periodic)
u0 = u;

[Dxx, Dyy, Dxc, Dyc, Dxb, Dyb, Dxf, Dyf, Dxyc] = ...
  diff2d_matrices(x1d, y1d, 0, 'p');

numsteps = 10;

v = u0(:);
```

```
% Time-stepping:
for m = 1:numsteps
  unew = v + k*(Dxx*v + Dyy*v);
  v = unew;
end

u2 = reshape(v, n, n2);

figure(2); clf;
imagesc(u2);
caxis([0 1])
colormap(gray)
axis equal, axis tight

result = [u  u2];
```

(c) The output is given below.



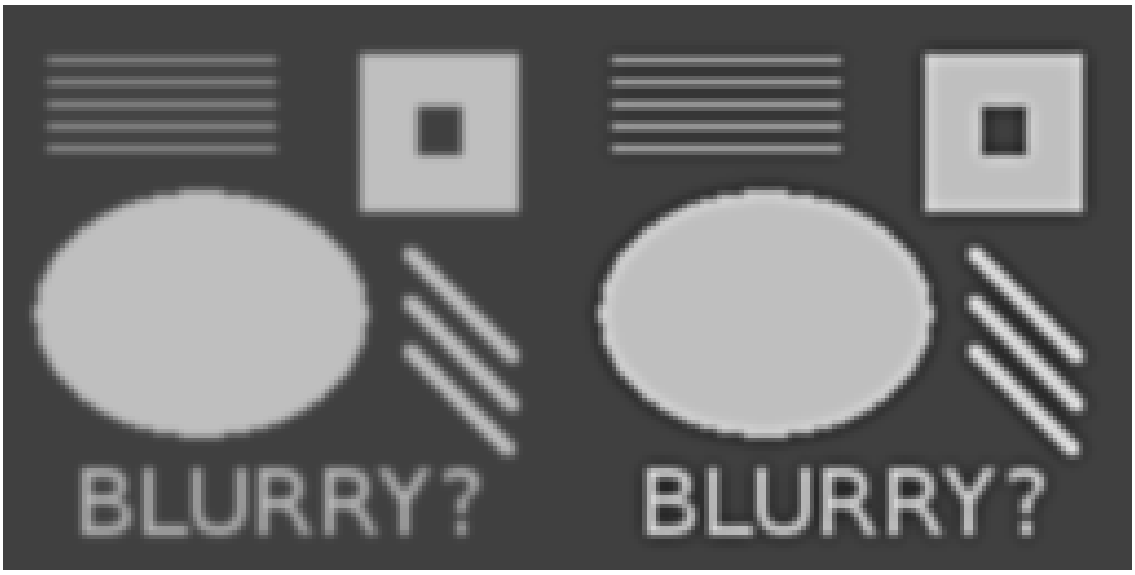
(d) As it can be seen below, the solution blows up. This is because we are taking too big steps, so we are not in the convergence zone. So the the solution start blowing up at boundary points and it spreads to the picture with each iteration.



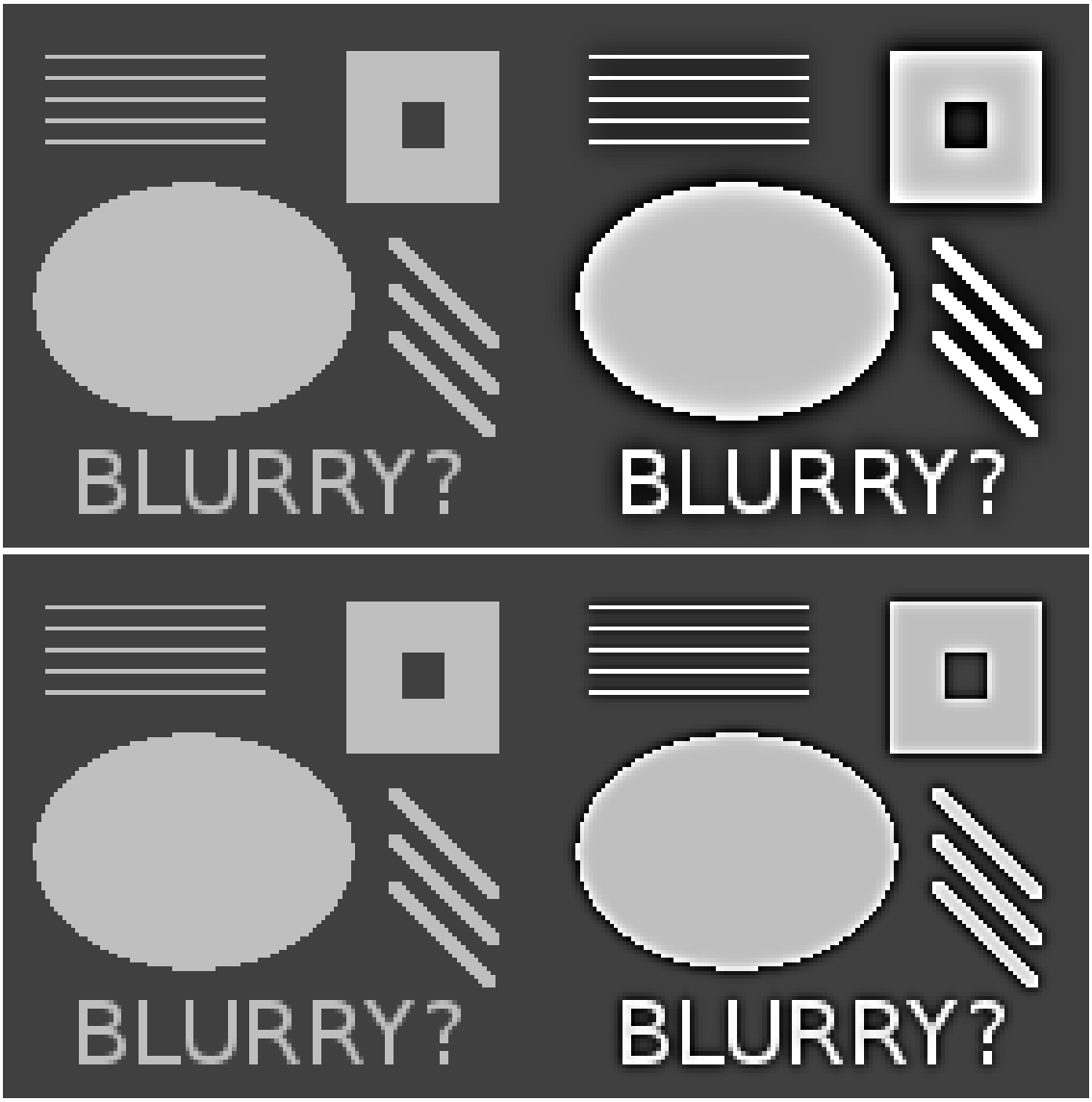
3. (a) The part of the code I modified is given below. The rest is the same as before:

```
...
% Time-stepping:
for m = 1:numsteps
  unew = v + k*(Dxx*v + Dyy*v);
  v = unew;
end
u2 = reshape(v, n, n2);

substr = u-u2;
unsharp_mask = u+ substr;
max_mask = max(unsharp_mask,[],'all');
min_mask = min(unsharp_mask, [],'all');


figure(2); clf;
imagesc(unsharp_mask);
caxis([0 1])
colormap(gray)
axis equal, axis tight


%% Output original and result side-by-side
% You could look at the resulting file with a web browser or image
% viewer
 result = [u  unsharp_mask];
 imwrite(result, 'result_masktestpatnoblur_numsteps50.png')
```

(b) max value is .8737 and the min value is .147

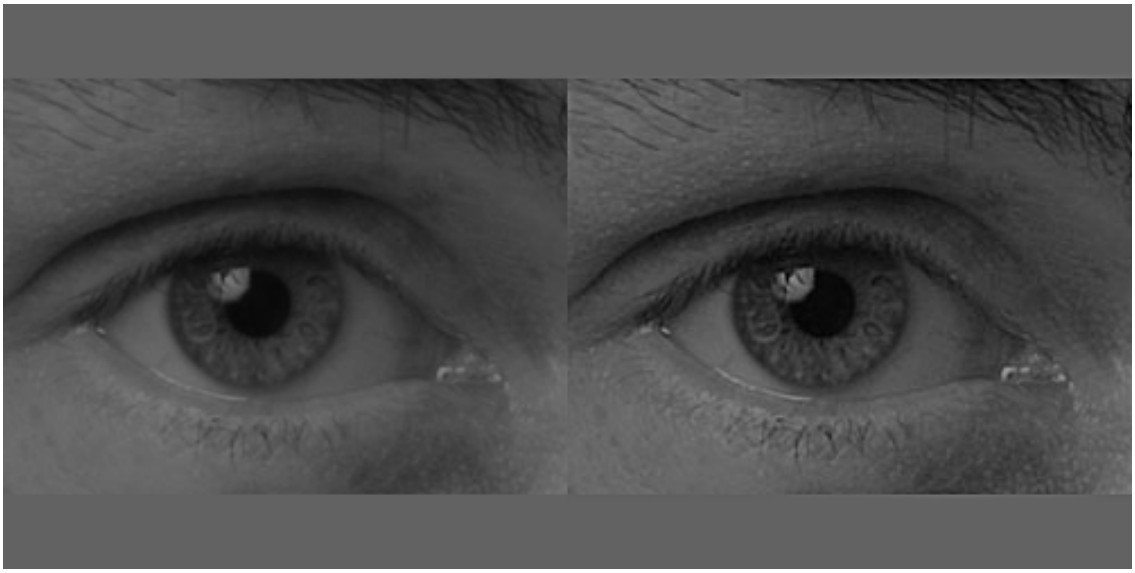


(c) The number of blurring steps, adjusts the intensity of blurring. So the higher this number is the more blurry the image gets, the colors start vanishing because of the effect of solving the heat equation which is also known as dissolution. The first image below shows the result after 50 numsteps and the second is default 10 numsteps.

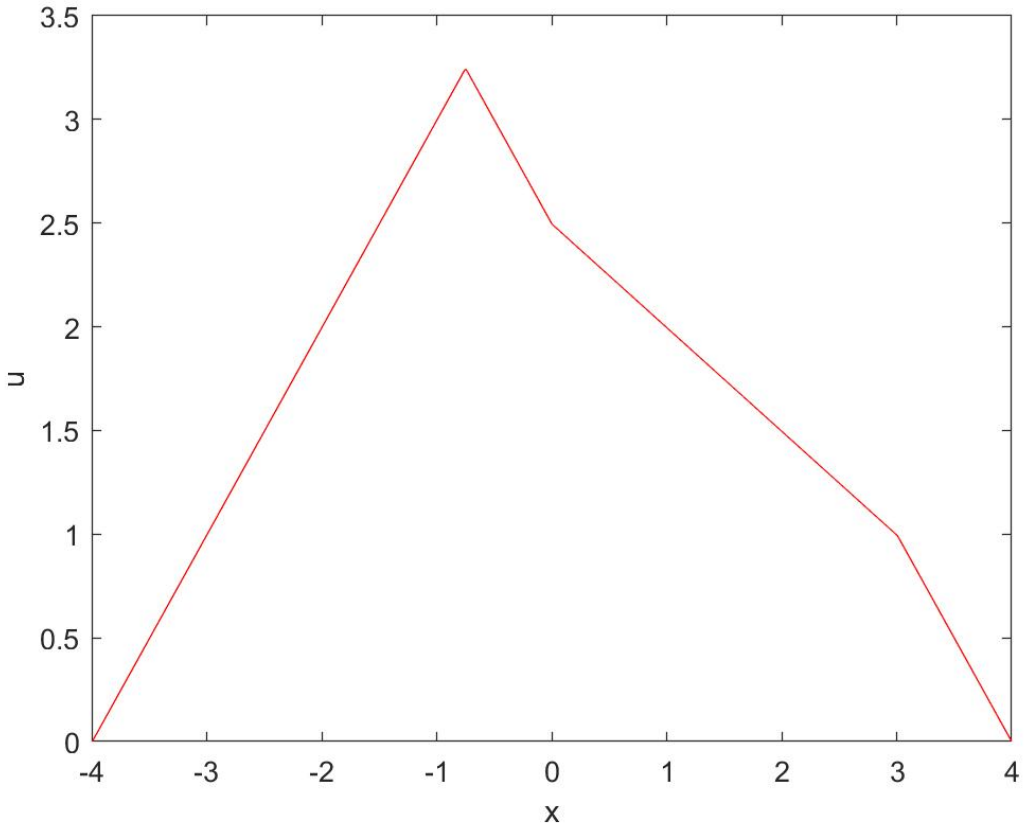


When we run the mask on the not blurred image, the edges where the original image is white gets whiter than white and the edges which are black get blacker than black. This can be seen by the minimum and maximum values: the maximum is 1.153 and the minimum is -.03

(d) Here is masking run on the eye image given:



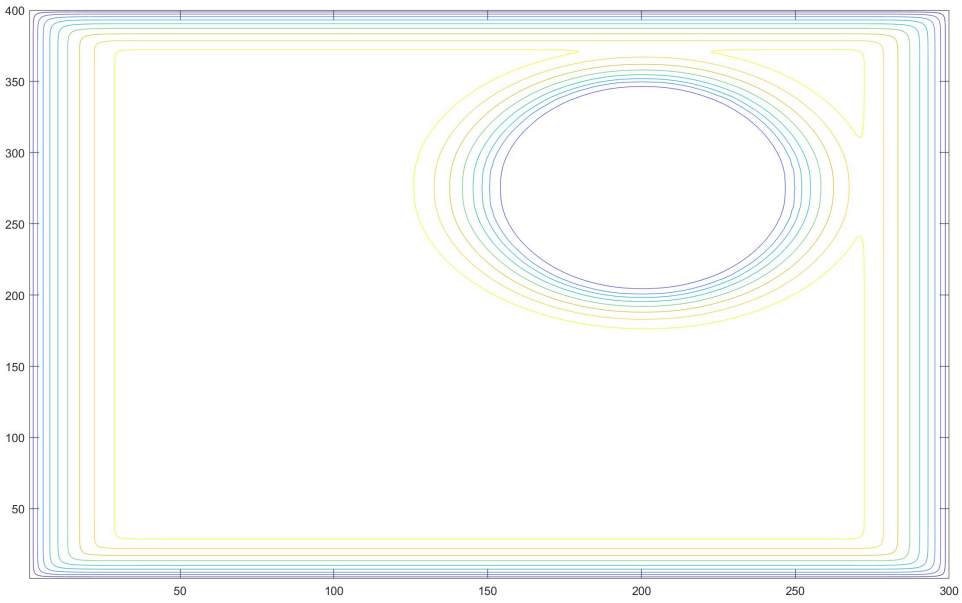
4. The solution in 1-D is given below with the code:

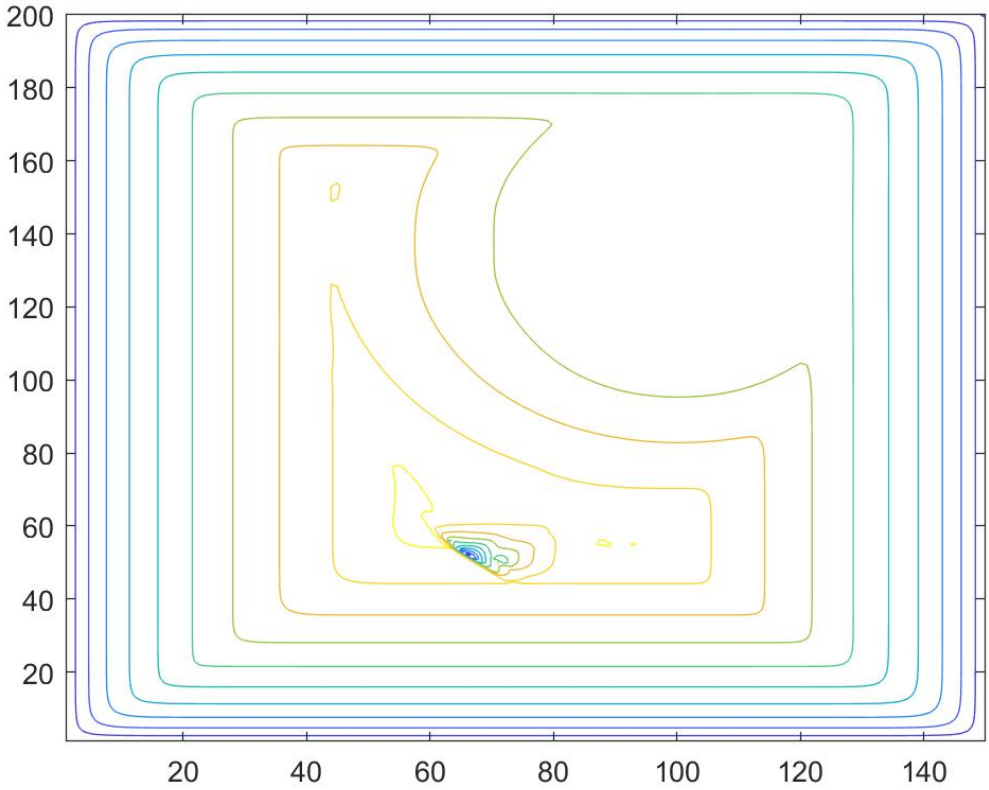



```
clc

J = 200;
h = 1/J;
x = (-4+h:h:4)';

f = zeros(size(x));
```

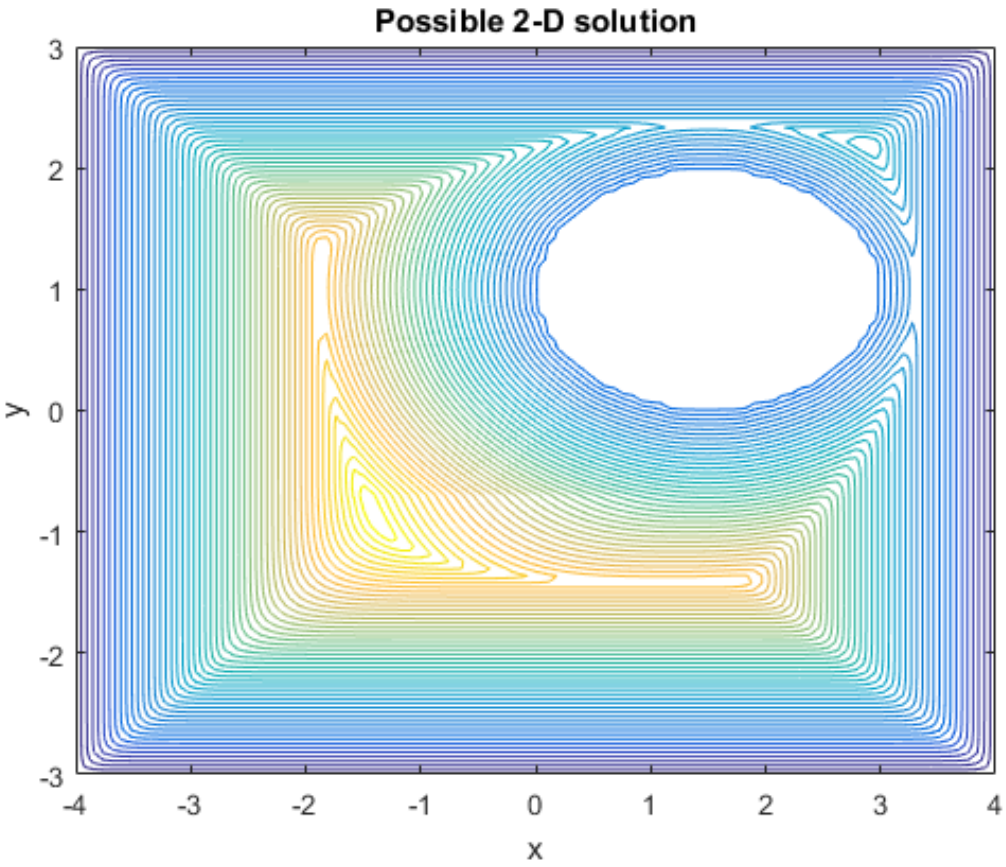
```
% I_c = (x >= -3) & (x <= -1) & (y >= -2) (y <= 1);
% I_w = 4/9*(x-3/2)^2 +(y-1)^2 <= 1;
% I_s = @(x) not(I_c) & not(I_w);

R = ones(size(x));

for i= 1:size(x)
    if (x(i) >= 0 && x(i) <= 3)
        R(i) = 1/2;
    end
end

iter = 10000;

f_new = zeros(size(f));

for j=1:iter
    for i= 2:(size(f)-1)
% I was setting the boundary conditions by hand then ignored them all
% together
%         if i == 2
%             f_new(i-1) = f(i)/2+h/2*(R(i-1)-f(i)/(2*h));
%         end
%         if i == size(f)-1
%             f_new(i+1) = f(i)/2+h/2*(R(i)-f(i)/(2*h));
%         end
        f_new(i) = (f(i+1)+f(i-1)) /2 + h/2* (R(i) - (abs(f(i+1)-f(i-1))) / (2*h));
        f = f_new;
    end
    if (mod(j, 50) == 0)
        plot(x, f, 'r-')
        xlabel('x'); ylabel('u')
        drawnow
    end
end
```

My code takes forever to run in 2-d for some reason. I discussed it with 2 computer science majors and they could not figure it out. I am including two pictures with iteration 100 and 130. And taking the last one from the discussions. I think that is where my graph is converging to.





The one below is taken from the discussions to show that my code is working in the right direction:



And my code:

```
clear all
clc

J = 25;
h = 1/J;
x = (-4+h:h:4)';
y = (-3+h:h:3)';
N = length(x);
M = length(y);
f = zeros(N,M);

I_c = @(i,j) (x(i) >= -3) & (x(i) <= -1) & (y(j) >= -2) & (y(j) <= 1);
I_w = @(i,j) 4/9*(x(i)-3/2)^2 +(y(j)-1)^2 <= 1;
I_s = @(i,j) not(I_c(i,j)) & not(I_w(i,j));

R = ones(N,M);

for i= 1:size(x)
    for j = 1:size(y)
        if I_w(i,j)
            R(i,j) = 1/2;
        end
    end
end

iter = 130;
```

```
f_new = zeros(size(f));
f_new(i-1,j-1) = 1;
f_new(end,end) = 1;

for s=1:iter
    for i= 2:(N-1)
        for j = 2:(M-1)
% I was setting the boundary conditions manually then ignored them all
% together with recommendation from professor Macdonald


            Dxc = (f(i+1,j)-f(i,j))/h;
            Dyc = (f(i,j+1)-f(i,j))/h;
            f_new(i,j) = (f(i-1,j)+f(i+1,j)+f(i,j-1)+f(i,j+1))/4 + ...
                h/4*(R(i,j) - I_c(i,j) *(abs(Dxc*f(i,j))+abs(Dyc*f(i,j))) - ...
                (I_s(i,j) | I_w(i,j)) * sqrt((Dxc*f(i,j))^2+(Dyc*f(i,j))^2));
            f = f_new;
        end
    end
    if (mod(s, 10) == 0)
        contour(f)
        drawnow
        display(s)
    end
end
```

So, the physical interpretation would be: assuming a person being in the center of the city how long it would take him to reach wherever he wants to go. The fastest way is the forest(Professor Macdonald said he made a mistake in setting up the question)